Chapter 4: Geometry and Zoning

The First Step

The first step in setting up a TART95 problem is to define its geometry by defining spatial zones in terms of the surfaces that bound these zones. Geometry refers to the boundary functions of a problem. Combining a number of boundary functions creates a unique and unambiguous volume called a zone. Presently TART95 uses three types of boundaries: planes, aligned quadratics and rotated quadratics. The quadratic boundaries describe surfaces that are either closed and completely bounded (e.g., a sphere, that it bounded by a closed surface at the radius of the sphere) or infinite (e.g. a cone, that extended to infinity, in **both** directions from its apex).

Planes are described by the equation,

$a(x0 - x) + b(y0-y) + c(z0-z) = 0$

For example, a plane perpendicular to the z axis at z = z0, is defined by the equation,

$z0 - z = 0, \quad a = b = 0, c = 1$

Aligned quadratics (aligned with x, y or z axis) are described by the equation,

$a(x0 - x)^2 + b(y0 - y)^2 + c(z0 - z)^2 = r^2$

For example, a sphere of radius r centered at the point x = x0, y = y0, z = z0, is defined by the equation,

$(x0 - x)^2 + (y0 - y)^2 + (z0 - z)^2 = r^2, \quad a = b = c = 1$, r = radius of the sphere

The aligned quadratic boundary function can be used to describe spheres, ellipsoids, cylinders, cones and hyperboloids of one or two sheets. This equation specifies conic sections that may be rotated symmetrically about the x, y or z axis or any axis parallel to the coordinate axes.

Rotated quadratics (initially aligned and then rotated using **surfp** or **srotate** input) are described by the equation,

$a(x0 - x)^2 + b(y0 - y)^2 + c(z0 - z)^2 +$
$d(x0 - x)(y0 - y) + e(x0 - x)(z0 - z) + f(y0 - y)(z0 - z) = r^2$

The cross terms in x, y, z result from the rotation of the initially aligned quadratic about its center (x0, y0, z0).

The procedures used to define and input the boundary function parameters, x0, y0, z0, a, b, c, etc., are described in detail in the section of this manual on **TART95 INPUT**. Here I will discuss how these parameters are actually used in calculations.

Coordinates of Particles

TART95 tracks neutrons and/or photons, both of which will be referred to as "particles", in the following discussion. A particle is defined by its spatial coordinates (x, y, z), its direction coordinates by direction cosines (alpha, beta, gamma), relative to the (x, y, z) axii, respectively, its speed (neutrons) or energy (gamma), and its time since it originated (t).

For scoring, TART95 uses these coordinates to define the track length of particles within spatial zones and within specified time intervals. The following sections will describe how TART95 tracks particles in space. First we will discuss how to best define the spatial coordinate system of your problems, to allow you to use as many features of TART95 as possible.

Preferred axis of orientation

TART95 uses general (x, y, z), 3-D geometry. However, it has a preferred axis of orientation = the z axis; this is strictly for historical reasons. For general 3-D problems there is no advantage to using any preferred axis of orientation. However, for problems that have one or more axii of symmetry there is a definite advantage when using TART95 to defining one axis of symmetry to be the z axis. When this is done: 1) you will be able to use all of the geometry keywords as input to define your space, 2) TART95 can analytically calculate the volume and mass of all zones that are symmetric about the z axis. For general 3-D problems or problems in which you use an axis of symmetry other than the z axis, TART95 cannot analytically calculate volumes and mass of zones; however, it can use Monte Carlo sampling to approximate the volume of zones.

Therefore it is recommended that if your problem has an axis of symmetry, you define this to be the z axis of your geometry. For example, if you have a number of concentric cylinders whose central axis are all oriented in the same direction, in general it is completely arbitrary what coordinate system you use and what direction you define this central axis to be. However, for use with TART95 it is recommended that you define this to be the z axis. Similarly if you have a number of parallel planes, it is recommended that you define them to be perpendicular to the z axis.

Zoning

TART95 uses 3-D combinatorial geometry. The geometry is 3-D in the sense that it always deals with 3-D volumes, as opposed to 2-D areas. Even if your geometry is inherently 1-D (e.g., planar or cylindrical) or 2-D (e.g., R-Z symmetric geometry), you should be aware that TART95 will track in 3-D and you MUST define your geometry

accordingly. For example, even if your geometry is 1-D cylindrical, you MUST be aware that defining a cylinder by input does not define a circle in a plane; it defines a 3-D cylinder extending to infinity in both directions along its axis. The geometry is combinatorial in the sense that each zone is defined by the combination of surfaces that bound that zone.

The coordinate system is (x, y, z) rectilinear space. Each problem is made up of a number of spatial zones. Each spatial zone is defined by the surfaces that bound the zone. In order to uniquely define zones, in addition to defining which surfaces bound each zone you must also define which side of each surface the zone is located on. It isn't enough to say that a zone is bounded by a plane at x0 = 5 cm, since the zone could be on either side of this plane. Is the zone all of the space with x < 5 cm, or all of the space with x > 5 cm? For example, if we have a simple problem only involving two concentric spheres, sphere 1 and 2, the problem could involve three spatial zones: 1) inside the inner sphere, 2) between the two spheres, 3) outside the outer sphere. In this case the definition of the three spatial zones should specify that the zones are: 1) **inside** the inner sphere, 2) **outside** the inner sphere and **inside** the outer sphere, 3) **outside** the outer sphere. Note, generally it is not sufficient to merely say a surface bounds a zone; you must also specify whether the zone is **inside** or **outside** the surface. For example, if in the previous example we had said that zone 2) is **inside** both spheres, rather than **outside** the inner sphere and **inside** the outer sphere, we would be describing a different spatial region. You might ask why would one possibly do such a thing. You will find that this is a typical error that one can very easily introduce into problems without realizing it. The concept of **inside** and **outside** surfaces is probably the most difficult thing for users of combinatorial geometry codes to get used to and completely understand; even experienced users will occasionally make errors. Therefore it is worth spending some time discussing it in detail.

TART95 surfaces are either linear or quadratic. Therefore all surfaces can be described by only considering two simple equations,

linear: $f(x, y, z) = a\,(x0 - x) + b\,(y0 - y) + c\,(z0 - z)$

quadratic: $f(x, y, z) = r^2 - [\, a\,(x0 - x)^2 + b\,(y0 - y)^2 + c\,(z0 - z)^2 \,]$

For any given spatial point (x, y, z) and surface with given parameters (x0, y0, z0, a, b, c, r) to determine whether the point is **inside** or **outside** the surface it is sufficient to insert these parameters into one of the above two equations and solve it. If the answer is **positive** the point is **inside** the surface, if **negative** the point is **outside** the surface, if **zero** the point is **on the surface**. This isn't as complicated as it sounds. Let's consider in detail all of the surfaces that you will have to deal with.

For simple closed surfaces, such as spheres, ellipsoids or cylinders the concept of **inside** and **outside** is fairly intuitive and easy to visualize. If you visualize a sphere, ellipsoid or cylinder you know what **inside** or **outside** means; you are either **inside** or **outside** the object, it's as simple as that. For example, for a sphere the equation is,

$f(x, y, z) = r^2 - [(x0 - x)^2 + (y0 - y)^2 + (z0 - z)^2]$

Note, $[(x0 - x)^2 + (y0 - y)^2 + (z0 - z)^2]$ is merely the square of the distance of the point (x, y, z) from the center of the sphere (x0, y0, z0). If this distance is less than the square of the radius of the sphere, r^2, the point is **inside** the sphere; conversely if this distance is larger than r^2 the point is **outside** the sphere.

For any spatial point (x, y, z) **inside** the sphere the result of solving this equation is **positive**, for any point **outside** the sphere it is **negative** and for any point **on the surface** of the sphere it is **zero**. So that for these surfaces the concept of **inside** and **outside** is simple to understand.

For open surfaces, such as cones and hyperboloids, it may not be as intuitively obvious what **inside** and **outside** mean. In these cases the surface will be with respect to a reference axis and will usually be rotationally symmetric about this reference axis. For example, for a circular cone that is rotationally symmetric about the z axis the equation is,

$f(x, y, z) = p(z - z0)^2 - [(x - x0)^2 + (y - y0)^2], a = b =1, p = -c > 0, r = 0$

At any z location $p(z - z0)^2$ is a constant, and $[(x - x0)^2 + (y - y0)^2]$ describes a circle in the z = constant plane. At z = z0, the radius of this circle is zero. For any other value of z the radius is positive and increasing when z is further from z0 (further in either direction from z0; the cone extends to infinity in z in both directions from the apex at z = z0). For any given (x, y, z) point, the point is **inside** the cone if at the given value of z the point is **inside** the circular cross section in the z = constant plane.

Any spatial point (x, y, z) that is closer to the reference axis than the surface, is **inside**, and any point that is further from the axis, is **outside**. Think of a cone or hyperboloid as a solid object between its axis of symmetry and its surface, e.g., for a cone visualize a drinking cup. Can you visualize whether a drop of water is **inside** or **outside** of your drinking cup? If you can, you will have no problem with these surfaces. If not, you can always solve the equation for the surface. For example, for a cone, pick any (x, y, z) point and solve this equation to define whether or not a point is **inside** or **outside** the surface.

For the last example, consider a plane perpendicular to the x axis at x = x0, the equation of the surface is,

$x0 - x = 0$

All spatial points (x, y, z) with x0 > x will be **inside** the surface (since x0 - x > 0), and all points with x0 < x will be **outside** the surface (since x0 - x < 0), and all points with x0 = x will be **on the surface**.

For planes you may prefer to think in terms of **below** and **above** instead of **inside** and **outside**. We encourage you not to do this. For simple aligned planes it may be easy for you to think of **above** and **below**, but for general rotated planes this can be very confusing. This is also true of general cones. The reason is that in these cases the equations are,

plane: $a (x0 - x) + b (y0 - y) + c (z0 - z) = 0$

cone: $-[a (x0 - x)^2 + b (y0 - y)^2 + c (z0 - z)^2] = 0 \quad r = 0,$
two coefficients, a, b, c > 0, one < 0

Note, that since r = 0, by changing the sign of all of the coefficients, a, b, c, we are defining exactly the same surface, but if we solve these equations the meaning of **inside** and **outside** based on the sign of the answer is completely reversed.

Fortunately only in the case of specifying general planes and quadratic surfaces will you be faced by this inconsistency in the sign of the coefficients defining the surface; in all others cases the program will automatically define the sign of all coefficients. When you do input parameters defining general planes or quadratics try to remember to define the coefficients a, b, and c, so that **inside** and **outside** make the most sense to you.

If you can understand the concept of **inside** and **outside** for planes, spheres, cylinders, cones, ellipsoids, hyperboloids, you are in, because those are all of the surfaces that you need understand when dealing with TART95. If you don't completely understand the concept don't worry about it that's the whole purpose of the **TARTCHEK** code, which will completely check your description of geometry and tell you whether or not you have made an error in specifying **inside** or **outside**. If you are not sure of the sign of a surface rather than worry about it, bang in anything, run **TARTCHEK**, and let this interactive graphics program immediately tell you whether or not you have made the right choice. If you have made a mistake you can then quickly change the sign of the surface and run **TARTCHEK** again to insure that you now have it right.

Tracking and Terminating Histories

TART95 will start from the current particle coordinates: space, direction, energy and time, and track the particle until the particle disappears (e.g., is absorbed) or is outside the region of interest - in terms of space, energy or time.

Elsewhere, how to limit tracking in energy and time are discussed. Here we will discuss how to limit space. TART95 will track particles in any zone that contains material and terminates the history if a particle enters an empty (void) zone. In a well defined geometry all interiors zones of interest to you MUST contain material, and all exterior zones MUST be empty (void), so that any particle entering any exterior zone will be terminated. By convention material number zero means a terminus zone, i.e., a void that TART95 will not track particles through. The default material for all zones is material number zero.

Therefore unless you explicitly assign a material to a zone, the zone is by default assumed to be a void, terminus zone.

Warning - this is the only way that TART95 knows how to limit spatial tracking, so that you must use care to insure that ALL interior zones contain material and your problem is surrounded by one or more exterior non-re-entrant zones which are empty (void); this point is discussed in detail below. If you leave any interior zone empty (void), this is not considered by the code to be an error, but all particles that enter such zones will have their history terminated - the zone will appear to be perfectly absorbing. Before starting any TART95 runs, it is strongly suggested that you use program **TARTCHEK**, to insure that you have no interior empty (void) zones.

Using Interior Void Zones

We will mention one case in which you may want to leave an interior zone void, thereby killing all particles that enter it; particles can never exit such a zone. You may want to do this if you have a geometrically complicated problem in which you want to vary the contents of a given zone, e.g., you might vary material, density, temperature, any parameters characterizing the contents of the zone. You can optimize the running time for a series of such calculations by first running a calculation with the zone void and involving many histories, only a small number of which will find their way to the zone. By specifying tally type 12 for the initially void zone the coordinates of all particles entering the zone will be written to a binary file. During subsequent runs you can change the geometry only by defining the composition of the zone and read the binary file to define a source incident on the zone. This source will then be tracked through your entire geometry and give you the answer you want. Note, if you do this for a series of subsequent runs each differing only in the contents of the one zone, the source incident on the zone will be exactly the same in each case (will be exactly correlated) which will tend to highlight the effects of any changes that you have made in the zone from one run to the next.

For example, consider a problem in which the probability of neutrons starting from a given source distribution finding a given zone is only 1/100, and you want to examine 10 changes in the composition of the zone. If we first perform an initial run involving a million source particles about 10,000 of them will find their way to the zone and these can all be used in subsequent calculations. We will then subsequently make a separate run for each of the 10 compositions; each of these runs will read the 10,000 histories from the binary file and each can contribute to the results that we want. In these 11 runs we will have tracked 1.1 million histories (a million in the initial run and 10 times 10,000 in the subsequent 10 runs). In a second approach we could make a separate run for each of the 10 compositions using 1,000,000 source particles to obtain the statistically same 10,000 histories that find their way to the zone. The second approach will take almost 10 times as long to run, and the source distributions at the surface of the zone for the 10 problems will not be completely correlated, as they are in the first approach.

Defining the Universe

When using TART95 you MUST define all space - the entire universe - not just the small region that you are really interested in. This isn't as difficult as it sounds. All it means is that in additional to the spatial region that is really of interest to you, you MUST surround your space by one or more closed, non-re-entrant surfaces, and define everything outside these surfaces (the rest of the universe) to be spatial regions, that do not contain any real material.

A non-re-entrant surface means that any particle crossing this surface can never re-enter the geometry in its current direction of travel. In this case the particle can never find its way back into the spatial region of interest, so that it can have no effect on the results, and can be ignored (its history can be terminated). For simple geometries, such as concentric spheres or cylinders it is easy to correctly define non-re-entrant surfaces completely surrounding the spatial region of interest. For example, for a series of concentric spheres with void outside the largest radius sphere, the largest radius sphere is a surrounding, non-re-entrant surface, since any particle crossing this surface can never find its way back into the spheres. Note, this is only true if there is vacuum outside the spheres; if there is air or anything else outside the spheres a particle could scatter and find its way back into the spheres. For more complicated geometries it may be more complicated. It is highly recommended that before actually running TART95 you use the interactive graphics code **TARTCHEK** to insure that you have correctly closed your geometry and do not have any void exterior or interior re-entrant zones.

Closing Your Geometry

You MUST define a finite volume spatial region that TART95 should track particles in. For problems that are inherently closed, e.g., a series of concentric spheres, this is obviously simple. In this case the finite spatial volume, is the entire volume enclosed by the largest radius sphere; the rest of the universe is everything outside this sphere.

In cases where none of the dimensions extend to infinity you can close your geometry by enclosing it inside a surrounding non-re-entrant surface or surfaces. For example, you can use a spherical surface that encloses all of the space that you are interested in. In some cases you may have to define more than one exterior zone in order to completely surround the space that you are actually interested in. For example, for a problem involving a number of concentric cylinders of finite length, you can close your geometry by defining empty (void) zones outside the largest radius cylinder and two planes, orthogonal to the cylinders, one at each end of the cylinders, with one empty (void) zone below the lower plane and another empty zone above the higher plane. In this case there are three exterior non-re-entrant zones all of which should to empty (void).

In both of the cases described above it is important that all space inside the bounding non-re-entrant surfaces contain material, i.e., are not empty (not avoid), and that all space outside these surfaces be empty (void), so that any particle entering any exterior zone will have its history terminated.

For other problems you may have to use reflecting surfaces to define a finite volume. For example, in a problem involving only a series of concentric cylinders extending to infinity along the axis of the cylinders, you can use reflecting planes that are orthogonal to the axis of the cylinders to define a finite volume space and yet still simulate transport in infinite length cylinders.

When using reflectors to simulate infinite dimensions or repeating spatial conditions, (e.g., a number of identical fuel elements), be sure that your reflectors truly simulate your geometry. For example, to simulate a 3-D infinitely repeating array of fuel elements, you can use reflectors to divide the infinite space into a number of square or hexagonal cells, since these shapes can exactly fill the entire 3-D space without any overlapping space. However, in this case you cannot use a reflecting cylinder, since you cannot arrange a series of cylinders in 3-D to completely fill the space without any spatial overlap.

Tracking in 3-D Geometry

In order to track (follow) particles in 3-D geometry TART95 has to be able to define: 1) what zone a source particle is initially located in, 2) what is the shortest distance along the particles direction of travel to any bounding surface of the zone the particle is currently located in, 3) if a particle reaches and crosses a bounding surface of the current zone, what spatial zone does it enter, 4) if a particle reaches and reflects from a bounding surface of the current zone, what is its new direction, 5) when scoring flux, rather than current, when crossing surfaces the code must be able to define the cosine of the direction of the particle relative to a vector normal to the surface (to allow scoring reciprocally weighted by this cosine to define flux). These are the only things that the code has to be able to define about the geometry in order to track in 3-D. The TART95 geometry package has been designed to be modular and virtually completely isolated from the remainder of the code. Only a few subroutines are required to completely define the geometry and allow particles to be tracked in 3-D geometry. Each of these routines is described below.

What Zone is a Particle Currently located in - WHATZONE

Starting from a source particle TART95 will cycle through all spatial zones in a problem and use the bounding surfaces for each zone to determine whether a particle in **inside** or **outside** a **zone**. The input describing each bounding surface for each zone defines whether the zone is **inside** or **outside** this bounding surface, as discussed above. A particle is **inside** a zone **if and only if** its spatial coordinates (x, y, z) satisfy **ALL** of the bounding surfaces. Starting from a source particle this search is optimized by the input for each source distribution indicating a most probable zone where the circular search of zones should start. **Warning** - the code will consider a particle to be in a zone the first time it finds any zone where the space point (x, y, z) satisfies all bounding surface conditions to indicate this point is within the zone. Therefore, the code can become confused if you have overlapping zones, where a given space point is in more than one zone. Before running

TART95 it is strongly suggested that you use **TARTCHEK** to insure that your geometry does not have overlapping zones.

When a particle reaches the boundary of a zone and crosses it, in order to determine which zone has now been entered, the code uses exactly the same logic as described above starting from sources. However, in this case the search is optimized by defining a most probable zone that will be entered when a particle leaves a given zone across a given surface. The user has the input option of explicitly defining the most probable zone entered using **bjp** input (not recommended) or the user can use **jb** input and allow the code to automatically do this optimization statistically, based on the first particle that crosses the surface once the problem starts running.

To determine if a particle is in a given zone the code has a list of bounding surfaces, defining the type of surface, e.g., plane, sphere, cylinder, cone, etc., the parameters of each surface, e.g., center and radius of a sphere, and an integer, +1 or -1, indicating whether the zone is **inside** or **outside** this surface. The only types of surfaces considered are,

linear: $f(x, y, z) = a\,(x0 - x) + b\,(y0 - y) + c\,(z0 - z)$

aligned quadratic: $f(x, y, z) = r^2 - [\, a\,(x0 - x)^2 + b\,(y0 - y)^2 + c\,(z0 - z)^2 \,]$

rotated quadratic: $f(x, y, z) = r^2 - [\, a\,(x0 - x)^2 + b\,(y0 - y)^2 + c\,(z0 - z)^2$
$d(x0 - x)(y0 - y) + e(x0 - x)(z0 - z) + f(y0 - y)(z0 - z)]$

If surface sign (+1 or -1) times f(x, y, z) is positive the point (x, y, z) is **inside** the zone; otherwise it is **outside** the zone. If f(x, y, z) is zero the point is on the boundary of the zone.

Starting from the most probable zone the code will test each zone in turn in a circular fashion, testing all bounding surfaces of each zone until it finds a bounding surface and sign that indicate the point (x, y, z) is outside the zone - in which case it will proceed to test the next zone - or, testing all bounding surfaces and signs indicates that the point is definitely within the zone, in which case the search is complete.

If after testing all zones the code finds that the point (x, y, z) is not in any zone, this is considered to be an error and execution will terminate. In order to avoid this problem during execution of TART95, it is strongly recommended that you use program **TARTCHEK** to insure that all spatial points are defined. Only after testing with **TARTCHEK** should you use TART95.

Distance to Boundary Calculation - HOWFAR

Once the code knows what zone a particle is located in, it must define the distance, and therefore time, to the next "event". Between events neutrons and photons move in their current direction of travel without interacting with the medium through which they are traveling. The next "event" could be: 1) a collision, 2) reaching a census time 3) reaching a spatial zone boundary, .

The distance to collision is sampled from an exponential distribution in accordance with the total cross section. The distance to census time is based on the difference between the census and current times divided by the speed of the particle. How to define the distance to a spatial zone boundary will be discussed here. The next "event" is defined by the smallest of the three possible distances: collision, census time, or boundary.

In order to define the distance to a boundary of a zone, consider a particle located at the spatial point (x, y, z) moving in the direction (alpha, beta, gamma), where these are the direction cosines with respect to the (x, y, z) axes. Between events the particles are considered to move in a straight line in their current direction of motion. The question is: how far does a particle have to travel in its current direction of motion in order to intercept a bounding surface of the zone that the particle is currently located in? We are only interested in the minimum, **positive**, distance to any of the bounding surfaces; we are not interested in zero distances, where the particle is on the boundary. In order to answer this question we have to calculate the distance to each and every one of the bounding surfaces of the zone that the particle is currently located in and keep track of the minimum, **positive** distance.

As described above, when we decide to advance a particle to the nearest bounding surface of a zone, the subsequent search by **WHATZONE** to define what zone is next entered is optimized by for each bounding surface of each zone, defining the most probable zone that the particle will enter when it leaves the current zone across any one of its bounding surfaces. For example, a given zone may have 30 bounding surfaces - for this zone each of these bounding surfaces defines what zone will be entered if a particle crosses it. Another zone may have 23 bounding surfaces, some of which may be the same as the bounding surfaces of the first zone - again for this zone each bounding surface defines which zone is entered if a particle crosses it. The new zone entered is not defined only once for each defined surface, but rather is defined for every zone that this surface bounds. Therefore, when we are calculating the minimum, **positive** distance to any bounding surface of the current zone, we need to keep track of both the minimum distance and what surface the particle will cross if we advance the particle to the surface.

Why are we only interested in **positive** distances? We are only interested in distances that will truly advance a particle in its current direction of travel to a point where it can leave a zone. This means that we are certainly not interested in negative distances. What about zero distances? How can advancing a particle zero distance allow it to leave a zone? It can't, and only considering **positive** distances avoids some rare, and yet still possible, events, where a particle is traveling on and parallel to one of the bounding surfaces, e.g., a source particle traveling exactly on the surface of a cylinder, and exactly parallel to the

axis of the cylinder - not at all a rare event if this is how the source distribution is defined. In cases such as this if we accepted the fact that the distance to the bounding cylinder is exactly zero and use this to advance particles they wouldn't move and the code could end up in an infinite loop. By realizing that since we are only interested in distances that will truly advance a particle to a point where it can leave a zone, we can ignore the zero distance solutions, i.e., advancing the particle zero distance cannot allow it to leave the zone.

In order to calculate the minimum, **positive** distance to any bounding surface of a zone, we have to consider the different types of bounding surfaces used by TART95. The only three types of surfaces considered are,

linear: $f(x, y, z) = a (x0 - x) + b (y0 - y) + c (z0 - z)$

aligned
quadratic: $f(x, y, z) = r^2 - [a (x0 - x)^2 + b (y0 - y)^2 + c (z0 - z)^2]$

rotated
quadratic: $f(x, y, z) = r^2 - [a (x0 - x)^2 + b (y0 - y)^2 + c (z0 - z)^2$
$d(x0 - x)(y0 - y) + e(x0 - x)(z0 - z) + f(y0 -y)(z0 - z)]$

In each case we have to define,

$f(x', y', z') = 0$

where,

$x' = x + R \text{ alpha}$
$y' = y + R \text{ beta}$
$z' = z + R \text{ gamma}$

and solve for R, the distance to the boundary. All this equation says is that if a particle is located at the point (x, y, z) and is moving in the direction (alpha, beta, gamma), it has to move a distance R in this direction in order to intercept the surface. This distance may be negative, zero, or positive; again, we are only interested in positive distances.

For linear surfaces we end up with a linear equation in R,

$C1 R + C0 = 0$

$R = -C0/C1$

$C1 = -[a \text{ alpha} + b \text{ beta} + c \text{ gamma}]$
$C0 = a (x0 - x) + b (y0 - y) + c (z0 - z)$

Note, C0 is the definition of the boundary function, f(x, y, z), so that if C0 is equal to zero , the particle is on the boundary; since we are only interested in **positive** distances we can ignore this solution. If C1 is equal to zero the particle is moving parallel to the surface and there is no solution that we are interested in.

For quadratic surfaces we end up with a quadratic equation in R,

$$C2\ R^2 + 2\ C1\ R + C0 = 0$$

$$R = [-\ C1 +/-\ Sqrt(C1^2 - C2\ C0)]/C2$$

For an aligned quadratic,

$$C2 = -[a\ alpha^2 + b\ beta^2 + c\ gamma^2]$$
$$C1 = [a\ alpha\ (x0 - x) + b\ beta\ (y0 - y) + c\ gamma\ (z0 - z)]$$
$$C0 = r^2 - [\ a\ (x0 - x)^2 + b\ (y0 - y)^2 + c\ (z0 - z)^2\ ]$$

For a rotated quadratic,

$$C2 = -[a\ alpha^2 + b\ beta^2 + c\ gamma^2 +$$
$$d\ alpha\ beta + e\ alpha\ gamma + f\ beta\ gamma]$$
$$C1 = [a\ alpha\ (x0 - x) + b\ beta\ (y0 - y) + c\ gamma\ (z0 - z) +$$
$$\{d(beta\ (x0 -x) + alpha\ (y0 -y)) +$$
$$e(gamma\ (x0 -x) + alpha\ (z0 -z)) +$$
$$f(gamma\ (y0 -y) + beta\ (z0 -z))\}/2]$$
$$C0 = r^2 - [\ a\ (x0 - x)^2 + b\ (y0 - y)^2 + c\ (z0 - z)^2 +$$
$$d(x0 - x)(y0 - y) + e(x0 - x)(z0 - z) + f(y0 -y)(z0 - z)]$$

For the rotated quadratic the first line of the definitions of C2, C1 and C0 are identical to the definitions for an aligned quadratic; the additional lines are due to the cross terms in x, y, z.

There is no real solutions if $C1^2 - C2\ C0$ is less than zero.

Note, C0 is the definition of the boundary function, f(x, y, z), so that if C0 is equal to zero , the particle is on the boundary; since we are only interested in **positive** distances we can ignore this solution. However, in this case we still have to consider the other root of the equation,

$$R = -2\ C1/C2$$

If C1 is equal to zero the only possible positive root is,

$$R = +\ Sqrt(-C0/C2)$$

If C2 is equal to zero the particle is moving parallel to the surface and one root is infinite, in which case we still have to consider the solution,

$R = -C0/(2\ C1)$

This case can occur when a particle is traveling parallel to the surface of a cone on one side of the apex of the cone, but it can still intercept the surface of the cone on the other side of the apex.

In addition to the limiting cases where one, or more, of the coefficients C2, C1, C0, are exactly zero, we must also consider cases where these coefficients approach these limits. We can consider two cases. The first case,

$C1^2 >> C2\ C0$

in which case we can expand the Sqrt using,

$Sqrt(1 + x) = 1 + x/2 - 3*x^2/2/2! + 5*x^3/2/3!$ .....

$x = C2\ C0/C1^2$

$R = [- C1 +/- Sqrt(C1^2 - C2\ C0)]/C2$
$= [- C1 +/- C1\ Sqrt(1 + x)]/C2,\quad x = -C2\ C0/C1^2$
$= [- C1 +/- C1\ (1 + x/2 - 3*x^2/2/2! + 5*x^3/2/3!$ .....$]/C2$

The two roots are,

$R = [- 2\ C1 - C1\ (x/2 - 3*x^2/2/2! + 5*x^3/2/3!$ .....$]/C2$, using the - radical
$R = [+ C1\ (x/2 - 3*x^2/2/2! + 5*x^3/2/3!$ .....$]/C2$, using the + radical

Note, as C0 approaches zero (x --> 0), the first of these yields the expected root, -2C1/C2, and as C2 approaches zero, the second of these yields the expected root, $+C1\ x/2/C2 = -C1\ C2\ C0/C1^2/2/C2 = -C0/(2\ C1)$. In general the above two solutions can be used not only in the limiting case where C2 or C0 is exactly zero, but also in cases where these limits are being approached.

The second case is,

$C1^2 << |C2\ C0|,\ C2\ C0 < 0$

Note, earlier it was stated that there are no real roots for $C1^2 - C2\ C0 < 0$. However when $C2\ C0 < 0$, $C1^2 - C2\ C0$ cannot be less than zero, and we must consider the case $C1^2 << |C2\ C0|$.

Again we can expand the Sqrt,

$x = C1^2/(c2\ C0)$

$R = [- C1 +/- Sqrt(C1^2 - C2\ C0)]/C2$
$= [- C1 +/- Sqrt(-C2\ C0)\ Sqrt(1 + x)]/C2, \quad x = -C1^2/C2\ C0 >= 0$
$= [- C1 +/- Sqrt(-C2\ C0)\ (1 + x/2 - 3*x^2/2/2! + 5*x^3/2/3! .....]/C2$

The two roots are,

$R = [- C1/C2 + Sqrt(-C0/C2)\ (1 + x/2 - 3*x^2/2/2! + 5*x^3/2/3! .....]$
$R = [- C1/C2 - Sqrt(-C0/C2)\ (1 + x/2 - 3*x^2/2/2! + 5*x^3/2/3! .....]$

Note, as C1 approaches zero (x ---> 0), the first of these yields the expected root, +Sqrt(-C0/C2). In general the above two solutions can be used not only in the limiting case where C1 is exactly zero, but also in cases where this limit is being approached.

In order to find the minimum, **positive**, distance to any bounding surface of the zone that a particle is currently located in, TART95 initializes the minimum distance to a number that is large compared to the dimensions of the system, e.g., 10^{60}. It then tests each surface in turn, updating the minimum distance, until all bounding surfaces have been tested.

In order to minimize the number of tests involved, and avoiding roundoff or undefined problems, due to division by zero or very small coefficients, instead of solving for the actual roots,

linear: $R = -C0/C1$

quadratic: $R = [- C1 +/- Sqrt(C1^2 - C2\ C0)]/C2$

where we have to divide by C1 or C2, TART95 solves for,

linear: $R' = R\ C1 = -C0$

quadratic: $R' = R\ C2 = [- C1 +/- Sqrt(C1^2 - C2\ C0)]$

and compares these roots to the current minimum distance, R_{min}, times C1 or C2, as appropriate. In each case only if R_{min} C1 or R_{min} C2 is positive and R' is positive and closer to zero, need we divide by C1 or C2 to define the new minimum distance to boundary. Note, these tests automatically handle the limiting cases where C1 or C2 are exactly zero, as well as the cases where these coefficients are very small and approaching zero. For example, if C1 is exactly zero, R_{min} C1 is exactly zero, and R' cannot be positive and closer to zero, so that all such roots are ignored. Similarly, when C1 is very small and approaching zero, R_{min} C1 will also be very small, and in order to be accepted R' would have to be positive and even smaller, in which case we can safely divide by C1 without worrying about overflow.

As actually implemented in TART95 these tests have to consider the sign (+ or -) of the coefficient C1 or C2, as appropriate. Even with this additional concern the resulting algorithm of very fast and efficient and avoids all roundoff and undefined number (due to division by zero) problems.

Advancing into a new zone

Once the minimum distance to a boundary has been defined and it has been decided to advance the particle to the boundary (as opposed to its having a collision or reaching census time), the particle is advanced in its current line of travel a distance R_{min} plus a small additional distance (the geometric uncertainty). By advancing the particle a distance R_{min} the particle will end up on the bounding surface. If we leave the particle there it will complicate the calculations when we next try to define what zone the particle is now in, e.g., if it is on a boundary between two zones, which zone is it "in"? (as described above). Therefore, we advance the particle an additional small amount to, hopefully, move the particle off the boundary and into a new zone. We can then efficiently use **WHATZONE** (as described above) to define what zone the particle has now entered. The default geometric uncertainty is 10^{-6} cm, which can be modified by user input using **sentl 10**. For most problems the default value is adequate. Only if you have zones which are very thin, and comparable to 10^{-6} cm or less, need you change the default; in this case failure to decrease the geometry uncertainty can allow particles to pass through very thin zones without ever "seeing" them. For very large zones there is generally no problem as long as the zones are not so large that adding 10^{-6} cm to the distance to boundary results in underflow and ignores the geometric uncertainty; if this happens the particle will end up on a boundary and **WHATZONE** can become confused when attempting to define the new zone the particle is located in. TART95 performs all calculations using REAL*8 (64 bit) arithmetic, which is accurate to about 16 digits. Therefore as long as the thickness of zones is small compared to 10^{+10} cm, the default geometric uncertainty is adequate. If you have larger zones you should increase the geometric uncertainty.

Reflection from a Zone - REFLECT

In TART95 any **zone** may be identified by input as a reflecting **zone**. It is important for the user to understand that with TART95 individual **zones**, not individual surfaces, are defined as reflecting. When a **zone** is identified as reflecting, particles crossing any bounding surface of the zone will be reflected from the surface of the zone. Input keywords **reflx**, **refly**, **reflz**, **reflgp**, and **reflq** identify **zones** as reflecting, like an x plane, y plane, z plane, general plane or quadratic surface, respectively. Since a zone may be bounded by more than one surface all of these keywords are treated as equivalent and whenever a particle enters a reflecting zone it will be reflected according to the type of surface it crossed (plane, sphere, cylinder, etc.), regardless of which input keyword was used to identify the zone as reflecting. However, in the output listing the code will identify how many particles where reflected from x, y and z planes. Therefore, if you are interested in this information it is suggested that you use the appropriate input keyword.

The method of entering a reflecting zone is exactly the same as described above, where a particle is advanced to the surface of a zone plus the geometric uncertainty and **WHATZONE** is then used to determine what zone the particle has entered. The material number in the new zone is then used to decide how to proceed: 1) if the material number is positive, transport continues in the zone in the current direction of travel, 2) if the material number is zero, the zone is empty (void) and the particle history is terminated (this should only be used for exterior, non-re-entrant zones), or 3) if the material number is negative, the particle is reflected from the zone. Whenever a reflection keyword is used as input to define a zone to be reflecting, TART95 will identify the zone as being reflecting by defining the material number assigned to the zone to be negative.

If the **zone** is reflecting the particle is first moved backwards along its current direction of travel by the geometric uncertainty, so that it is exactly on the surface that it crossed to enter the zone. The particle is then reflected from this surface by defining three new direction cosines (alpha, beta, gamma). It is then advanced in its new direction by the geometric uncertainty, to move it off the boundary to the inside of a zone. **WHATZONE** is then used to define what zone the particle is now located in and the cycle of tracking, terminating, or reflecting the particle continues.

In order to reflect particles the code first defines the normal to the surface. We will only be interested in points (x, y, z) that are located on the surface and we will define an outward directed unit vector normal to the surface. The types of surfaces we must consider are,

linear: $f(x, y, z) = a\,(x0 - x) + b\,(y0 - y) + c\,(z0 - z)$

aligned
quadratic: $f(x, y, z) = r^2 - [\, a\,(x0 - x)^2 + b\,(y0 - y)^2 + c\,(z0 - z)^2 \,]$

rotated
quadratic: $f(x, y, z) = r^2 - [\, a\,(x0 - x)^2 + b\,(y0 - y)^2 + c\,(z0 - z)^2$
$d(x0 - x)(y0 - y) + e(x0 - x)(z0 - z) + f(y0 - y)(z0 - z)]$

For each type of surface for any point (x, y, z) on the surface, $f(x, y, z) = 0$, and we can write,

linear: $a\,x + b\,y + c\,z = a\,x0 + b\,y0 + c\,z0 = r$

aligned
quadratic: $a\,(x0 - x)^2 + b\,(y0 - y)^2 + c\,(z0 - z)^2 = r^2$

rotated
quadratic: $a\,(x0 - x)^2 + b\,(y0 - y)^2 + c\,(z0 - z)^2 +$
$d(x0 - x)(y0 - y) + e(x0 - x)(z0 - z) + f(y0 - y)(z0 - z)] = r^2$

The components of the surface normal to the x, y, z axii are defined by taking the three partial derivatives of the boundary function with respect to x, y and z.

In the linear case,

$d(f)/d(x) = a$
$d(f)/d(y) = b$
$d(f)/d(z) = c$

In the aligned quadratic case,

$d(f)/d(x) = 2a(x0 - x)$
$d(f)/d(y) = 2b(y0 - y)$
$d(f)/d(z) = 2c(z0 - z)$

and in the rotated quadratic case,

$d(f)/d(x) = 2a(x0 - x) - [d(y0 - y) + e(z0 - z)]$
$d(f)/d(y) = 2b(y0 - y) - [d(x0 - x) + f(z0 - z)]$
$d(f)/d(z) = 2c(z0 - z) - [e(x0 - x) + f(y0 - y)]$

A normalized vector is defined by normalizing each component by dividing by,

$$\mathrm{Sqrt}[\{d(f)/d(x)\}^2 + \{d(f)/d(y)\}^2 + \{d(f)/d(z)\}^2]$$

For the linear case the normalization is,

$$\mathrm{Sqrt}[a^2 + b^2 + c^2]$$

For the aligned quadratic case it is,

$$2\ \mathrm{Sqrt}[a^2(x0 - x)^2 + b^2(y0 - y)^2 + c^2(z0 - z)^2]$$

For the rotated quadratic case it is,

$$\mathrm{Sqrt}[\{2a(x0 - x) - [d(y0 - y) + e(z0 - z)]\}^2 + \{2b(y0 - y) - [d(x0 - x) + f(z0 - z)]\}^2 + \{2c(z0 - z) - [e(x0 - x) + f(y0 - y)]\}^2]$$

In the linear case the equation of the outward directed unit vector normal to the surface is,

$$N = i\,e + j\,f + k\,g$$

where,

$R = \text{Sqrt}[a^2 + b^2 + c^2]$, (normalization)
$e = a/R$
$f = b/R$
$g = c/R$

For example, for a plane perpendicular to the x axis (**xplane**), a = 1, b = c = 0, and the outward direct unit vector normal to the surface is parallel to the x axis,

$R = 1$
$e = 1$
$f = 0$
$g = 0$

In the aligned quadratic case the equation of the outward directed unit vector normal to the surface is,

$N = i\,e + j\,f + k\,g$

where,

$R = \text{Sqrt}(a^2(x0 - x)^2 + b^2(y0 - y)^2 + c^2(z0 - z)^2)$, (normalization)
$e = a\,(x - x0)/R$
$f = b\,(y - y0)/R$
$g = c\,(z - z0)/R$

For example, for a sphere, r = the radius of the sphere, a = b = c = 1,

$R = \text{Sqrt}((x0 - x)^2+(y0 - y)^2+(z0 - z)^2)=r$ (radius of the sphere)
$e = (x - x0)/R$
$f = (y - y0)/R$
$g = (z - z0)/R$

In the rotated quadratic case the equation of the outward directed unit vector normal to the surface is,

$N = i\,e + j\,f + k\,g$

where,

$$R = \text{Sqrt}[\{2a(x0 - x) - [d(y0 - y) + e(z0 - z)]\}^2 + \{2b(y0 - y) - [d(x0 - x) + f(z0 - z)]\}^2 + \{2c(z0 - z) - [e(x0 - x) + f(y0 - y)]\}^2], \text{ (normalization)}$$

$e = \{2a\,(x - x0) - [d(y0 - y) + e(z0 - z)]\}/R$
$f = \{2b\,(y - y0) - [d(x0 - x) + f(z0 - z)]\}/R$

g = {2c (z - z0) -[e(x0 -x) + f(y0 -y)]}/R

For a particle located at a point (x, y, z) on the surface, moving with direction cosines (alpha, beta, gamma),

P = i alpha + j beta + k gamma

In order to properly reflect the particle we must define new direction cosines (alpha', beta', gamma'),

Q = i alpha' + j beta' + k gamma'

such that,

P x N = Q x N

where N is the vector normal to the surface. Solving for the new direction cosines we find,

alpha' = alpha - 2 e Cos(theta)
beta' = beta - 2 f Cos(theta)
gamma' = gamma - 2 g Cos(theta)

where (e, f, g) are the components of the unit normal vector and,

Cos(theta) = P N = alpha e + beta f + gamma g

is the cosine of the angle between the outward directed unit vector normal to the surface and the direction of travel of the particle. Note, this cosine is calculated by the code not only to reflect particles, but also when scoring flux crossing surfaces (which must by reciprocally weighted by this cosine).

For example, again considering a plane perpendicular to the x axis, (**xplane**), a = 1, b = c = 0, and the outward direct unit vector normal to the surface is parallel to the x axis,

R = 1
e = 1
f = 0
g = 0

Cos(theta) = alpha

alpha' = alpha - 2 alpha = - alpha
beta' = beta
gamma' = gamma

the direction cosine with respect to the x axis (alpha) is reversed and the other two direction cosines are unchanged.

As a second example, again consider a sphere, r = the radius of the sphere, and a = b = c = 1,

R = Sqrt((x0 -x)^2+(y0 - y)^2+(z0 - z)^2)=r (radius of the sphere)
e = (x - x0)/r
f = (y - y0)/r
g = (z - z0)/r

Cos(theta) = [alpha (x - x0) + beta (y-y0) + gamma (z - z0)]/r

alpha' = alpha - 2 (x-x0)[alpha (x-x0) + beta (y-y0) + gamma (z-z0)]/r^2
beta' = beta - 2 (y-y0)[alpha (x-x0) + beta (y-y0) + gamma (z-z0)]/r^2
gamma'= gamma- 2 (z-z0)[alpha (x-x0) + beta (y-y0) + gamma (z-z0)]/r^2

In this case the results are more complicated than in the case of an x plane, but still they are fairly easy to interpret. For example, for a point on an axis,

1) y = y0, z = z0, (x - x0)^2 = r^2
   alpha' = alpha - 2 alpha = -alpha
   beta' = beta
   gamma' = gamma

2) x = x0, z = z0, (y - y0)^2 = r^2
   alpha' = alpha
   beta' = beta - 2 beta = -beta
   gamma' = gamma

3) x = x0, y = y0, (z - z0)^2 = r^2
   alpha' = alpha
   beta' = beta
   gamma' = gamma - 2 gamma = -gamma

In these three cases the reflection is identical to that obtained reflecting off of a plane perpendicular to the x, y or z axis, respectively, where one of the direction cosines is reversed and the other two are unchanged.

Consistency between WHATZONE and HOWFAR

Both **WHATZONE** and **HOWFAR** calculate the boundary function,

linear: f(x, y, z) = a (x0 - x) + b (y0 - y) + c (z0 - z)

quadratic: f(x, y, z) = r^2 - [a (x0 - x)^2 + b (y0 - y)^2 + c (z0 - z)^2]

rotated
quadratic: $f(x, y, z) = r^2 - [a (x0 - x)^2 + b (y0 - y)^2 + c (z0 - z)^2$
$d(x0 - x)(y0 - y) + e(x0 - x)(z0 - z) + f(y0 -y)(z0 - z)]$

WHATZONE calculates f(x, y, z) to determine if the point (x, y, z) is inside a zone and **HOWFAR** calculates f(x, y, z) to define C0 of the linear or quadratic equation defining the distance to the boundary. Since we are dealing with computers of finite accuracy it is very important that both **WHATZONE** and **HOWFAR** perform these calculations in exactly the same order to define f(x, y, z). Failure to do this can result in inconsistent results where **WHATZONE** says a point is slightly inside a zone and **HOWFAR** says it's outside and cannot correctly calculate a positive distance to a boundary.

Optimization of Geometry routines

TART95 spends a fair fraction of its time tracking particles using the geometry routines **WHATZONE**, **HOWFAR**, and **REFLECT**, so that it is important to optimize the running time of these routines. In the above discussion only two types of boundary surfaces where described: planar and quadratic. As actually coded in TART95 each type of surface that can be defined by input parameters is explicitly treated as a case in order to minimize the number of operations that must be performed.

In the case of planes, rather than always considering the general case,

$f(x, y, z) = a (x0 - x) + b (y0 - y) + c (z0 - z)$

TART95 takes advantage of the fact that in many cases some of the coefficients (a, b, c) are zero and others are unity. For example, in the case of an **xplane**, a = 1, b = c = 0, and the code only has to consider the much simpler equation,

$f(x, y, z) = x0 - x$

In **WHATZONE** only one subtraction is required to define the equation we need to decide whether or not a particle is inside a zone, based on this surface. In **HOWFAR** the distance to boundary is,

$R = (x0 - x)/alpha$

and in **REFLECT** all that need be done is reverse the sign of the direction cosine with respect to the x axis, alpha.

Similarly, in the case of quadratics, rather than always considering the general case,

$f(x, y, z) = r^2 - [a (x0 - x)^2 + b (y0 - y)^2 + c (z0 - z)^2]$

TART95 takes advantage of the fact that in many cases some of the coefficients (a, b, c) are zero and others are unity. For example, in the case of a cylinder parallel to the x axis (**cylx**), a = 0, b = c = 1, r = the radius of the cylinder, and the code only has to consider the much simpler equation,

$$f(x, y, z) = r^2 - [(y0 - y)^2 + (z0 - z)^2]$$

WHATZONE, **HOWFAR** and **REFLECT** all start from this simpler form to speed up calculations.

Optimizing Geometry Input Parameters

By understanding how TART95 will use your description of geometry you can optimize your input parameters. I should stress that none of the following considerations are necessary in order to successfully use TART95. These are merely general guidelines that should be considered when defining your geometry in order to minimize TART95 running time. However, in designing the description of your geometry you should consider not only TART95 running time, but also the clarity of the description as far as you being able to understand it; generally your understanding is more important than optimizing running time. A few points to consider,

Each zone is assigned a number to identify it, i.e., an integer from 1 to the maximum number of zones allowed (currently 1000). In principle you can assign any number to any zone. In practice you can optimize your input by assigning zone numbers starting at 1 and incrementing by 1 sequentially to the number of zones in your problem. When a particle leaves a zone and enters another zone **WHATZONE** is used to search for and define which zone has been entered. This search is done in a circular fashion starting at the most probable zone and continuing to search all zones until the correct zone is found. During this search each zone number is tested in order to determine, first if this zone number is used in the problem, and second, if the zone number is being used, whether or not the particle is located in this zone. Therefore, if you have a problem involving only ten zones, you can number them 1 through 10, or if you wish you can number them 100, 200, 300 ....., 1000; it's your choice. But you should understand that the latter choice will slow down the calculation, since **WHATZONE** will have to search through 1000 zones, in 990 cases merely to find out that the zone number is not used, and only in 10 cases actually testing to determine whether or not the particle is in the zone.

A second consideration in assigning zone numbers, is to realize that due to the circular search of zones in **WHATZONE** the search can be further optimized to defining zone numbers in what you consider to be the most probable direction of particle flow, or transport, i.e., generally from the source to the outer extremities of your geometry. For example, if you have a point source at the center of a number of concentric spheres, the most probable direction of flow is from inner most to outer most radius sphere, so that you should number the zones from 1 to the number of spheres, from smallest to largest radius. For simply connected geometry, such as a series of concentric spheres where a

particle leaving one zone can only enter one other zone across a given surface, the order of zone numbers is not important, since TART95 will automatically define the most probable zone entered, in order to optimize the search in **WHATZONE**. Only in complicated truly 3-D geometry will the order of assignment of zone number be important.

In defining the surfaces bounding each zone, the order of the surface is completely arbitrary. For example, for a cylinder of finite length, you can define a zone to be: 1) inside the cylinder, 2) above a plane at the bottom of the cylinder, and 3) below a plane at the top of the cylinder. In your input you can define these three bounding surfaces in any order. You can optimize your input by ordering the input surface, first by the most restrictive in terms of volume, and next in terms of the simplicity of the surface, e.g., calculations involving simple aligned planes are the fastest, rotated general quadratics are the most time consuming. For example, if you have a number of closely spaced concentric cylinders that are very long, the most restrictive surfaces in terms of volume will be two closely spaced cylinders defining the inner and outer radius of the zone, so that your input should define the two bounding cylinders first. However, if you have a cylinder divided by closely spaced planes along its axis, the most restrictive surfaces in terms of volume will be two closely spaced planes, so that your input should define the two planes first. In either of the two cases discussed here the idea is to allow **WHATZONE** to reject zones by testing the minimum possible number of bounding surfaces before deciding the particle is not in the zone and allowing it to move on to test the next zone. For example, for a series of zones defined by cylinders and closely spaced planes, **WHATZONE** need only test planes until it finds a zone bounding by two planes containing the point (x, y, z); only then need it test cylindrical surfaces.